Apple Assembly Line

$2.40

Volume 8 -- Issue 6 March, 1988

## Something Personal

I had another birthday last weekend. Now my age is a perfect square again, for the seventh time. Another 15 years and it will be square again. At most I can expect two or three more square ages. Numbers like these are interesting to me. Moses said, after recalling the brevity and frailty of human life, "So teach us to number our days, that we may apply our hearts unto wisdom."

Certainly we all should echo that prayer. We want our years to count for something. Solomon said, "The fear of the LORD is the beginning of knowledge: but fools despise wisdom and instruction." David said, "The fool has said in his heart, 'There is no God'." James said, "If any of you lack wisdom, let him ask of God, who gives to all men liberally, and upbraids not; and it shall be given him." Paul said, "In Jesus Christ are hidden all the treasures of wisdom and knowledge."

And I say, the longer I live, the more I find that God is there, and faithful to his Word. The better part of wisdom is trusting the truly trustworthy. Solomon says it best: "Trust in the LORD with all your heart, and lean not unto your own understanding. In all your ways acknowledge him, and he shall direct your paths."

As I enter my jubilee year, I re-commit myself to doing just that.

## Peeking Inside AppleWorks 1.3, Part 4: Application Overlay Manager............Bob Sander-Cederlof

AppleWorks is a lot bigger than a normal Apple. The 6502 microprocessor is restricted to 64K of memory address space, and the fraction of this left to a ProDOS-based program is normally less than 48K. The AppleWorks applications use large areas of memory for the active data, so the amount left for programs is very small. So how does it work? By using overlays.

The file on an AppleWorks Program disk named APLWORKS.SYSTEM (which we have been analyzing over the last four months) contains subroutines which stay in RAM regardless of what activity you are performing. After starting up AppleWorks, this code resides in RAM from $1000 up to $2Exx. RAM from $800 to $FFF is used for various variables and buffers. RAM from $3000 up to $BEFF is used for application programs and data.

All of the rest of the code for the three applications is on the two files named SEG.M0 and SEG.M1. These two files together contain 43 small segments of code, designed to be loaded into RAM only when needed. The first 15 segments appear to me to be the data base application; the next 8, the word processor; and the rest, the spreadsheet. At the beginning of each file is an "index" which tells where in the file each program segment begins. The article "Dissecting AppleWords SEG.M0 and SEG.M1 Files" later in this same issue goes into detail about the structure of the index.

This month we will look in detail at the code inside AppleWorks which loads in the various overlays. Since it happens to be very near the beginning of APLWORKS.SYSTEM, I am also listing the JMP vector that starts at $1000.

Since each of the 43 overlays will need to use various subroutines from APLWORKS.SYSTEM, the author of AppleWorks needed a straightforward way for them to know their addresses. Woz's monitor is a good example of what happens when you do not provide such a method.

Back in 1977 we found hundreds of neat entry points in that tiny little ROM, all at very specific addresses. We used them flagrantly, to save RAM for better uses. Then Apple started taking routines out, moving others, until they finally printed a list of the 110 or so they will continue to support. There is no easy system to using these entry points, because Woz originally coded them with the idea of squeezing the most function into the smallest amount of memory.

The Apple IIgs monitor, on the other hand, is a good example of what happens when you go overboard in trying to provide a calling system. After acquiring over ten pounds of documentation, I still am only dimly understanding all the ins and outs of the toolboxes. I know it all starts by loading a 16-bit coded number into the X-register, and doing a JSL $E10000 command. Parameters are passed on the stack, and an error code is returned in the A-register. All is done very systematically, very cleanly, very macintoshly, but not very efficiently. The toolbox calls must be done in full 16-bit mode, cannot use the registers to pass data, eat up many machine cycles getting to the actual tool and back again, and do require the use of the A- and X-registers. Still, it may be the best way to create, organize, and control an open-ended set of tools in a machine like the IIgs.

ProDOS-8 MLI gives an example of another method, in which a single entry point processes all calls. The byte following the JSR to that entry point

| | | | |
|---|---|---|---|
| S-C Macro Assembler Version 2.0 | | DOS $100, ProDOS $100, both for $120 | |
| Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners | | $20 | |
| ProDOS Upgrade Kit for Version 2.0 DOS owners | | $30 | |
| Cross Assemblers for owners of S-C Macro Assembler<br>(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, Mitsubishi 50740, others) | | $32.50 to $50 each | |
| Source Code of any S-C Macro Assembler or Cross Assembler | | each, additional $100 | |
| S-C DisAssembler (ProDOS only) | | without source code $30, with source $50 | |
| RAK-Ware DISASM (DOS only) | | without source code $30, with source $50 | |
| ProVIEW (ProDOS-based disk utility program) | | $20 | |
| Full Screen Editor for S-C Macro (with complete source code) | | $49 | |
| S-C Cross Reference Utility | | without source code $20, with source $50 | |
| S-C Word Processor, both DOS & ProDOS, both 40- & 80-columns, with complete source code | | $50 | |
| DP18 and DPFP, double precision math for Applesoft, including complete source code. | | $50 | |
| ES-CAPE (Extended S-C Applesoft Program Editor), including Version 2.0 With Source Code | | $50 | |
| ES-CAPE Version 2.0 and Source Code Update (for Registered Owners) | | $30 | |
| Bag of Tricks 2 (Quality Software) | ($49.95) | $45 | * |
| S-C Documentor (complete commented source code of Applesoft ROMs) | | $50 | |
| Copy II Plus (Central Point Software) | ($39.95) | $30 | * |
| AAL Quarterly Disks | | each $15, or any four for $45 | |

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

| | | | |
|---|---|---|---|
| Vinyl disk pages, 6"x8.5", hold two disks each | | 10 for $6 | * |
| Diskette Mailing Protectors (hold 1 or 2 disks)<br>(Cardboard folders designed to fit 6"X9" Envelopes.) | | 40 cents each<br>or $25 per 100 | * |
| Envelopes for Diskette Mailers | | 6 cents each | |
| Sider 20 Meg Hard Disk, includes controller & software | ($695) | $550 | + |
| Sider 40 Meg Hard Disk, includes controller & software | ($995) | $860 | + |
| Minuteman 250 Uninterruptible Power Supply | ($359) | $320 | + |
| Minuteman 300 Uninterruptible Power Supply | ($549) | $490 | + |
| 65802 Microprocessor, 4 MHz (Western Design Center) | | $25 | * |
| quikLoader EPROM System (SCRG) | ($179) | $170 | * |
| PROmGRAMER (SCRG) | ($149.50) | $140 | * |

| | | | | |
|---|---|---|---|---|
| "Exploring the Apple IIgs" | Gary B. Little | ($22.95) | $21 | * |
| "Apple IIgs Technical Reference" | Michael Fischer | ($19.95) | $19 | * |
| "65816/65802 Assembly Language Programming" | Michael Fischer | ($21.95) | $20 | * |
| "Programming the 65816" | David Eyes & Ron Lichty | ($22.95) | $21 | * |
| "Programming the Apple IIgs in C and Assembly Language" | Mark Andrews | ($18.95) | $18 | * |
| "Apple //e Reference Manual" | Apple Computer | ($24.95) | $23 | * |
| "Apple //c Reference Manual" | Apple Computer | ($24.95) | $23 | * |
| "ProDOS-8 Technical Reference Manual" | Apple Computer | ($29.95) | $27 | * |
| "ProDOS-16 Technical Reference Manual" | Apple Computer | ($29.95) | $27 | * |
| "Apple IIgs Firmware Reference" | Apple Computer | ($24.95) | $23 | * |
| "Apple IIgs Hardware Reference" | Apple Computer | ($24.95) | $23 | * |
| "Apple IIgs Toolbox Reference, Volume 1" | Apple Computer | ($26.95) | $24 | * |
| "Apple IIgs Toolbox Reference, Volume 2" | Apple Computer | ($26.95) | $24 | * |
| "Apple IIgs Programmer's Introduction" | Apple Computer | ($32.95) | $30 | * |
| "ProDOS Inside and Out" | Dennis Doms & Tom Weishaar | ($16.95) | $16 | * |
| "Beneath AppleProDOS" | Don Worth & Pieter Lechner | ($19.95) | $18 | * |
| "Beneath Apple DOS" | Don Worth & Pieter Lechner | ($19.95) | $18 | * |
| "Inside the Apple //c" | Gary B. Little | ($19.95) | $18 | * |
| "Inside the Apple //e" | Gary B. Little | ($19.95) | $18 | * |
| "Understanding the Apple //e" | Jim Sather | ($24.95) | $23 | * |
| "Understanding the Apple II" | Jim Sather | ($22.95) | $21 | * |
| "Apple II+/IIe Troubleshooting & Repair Guide" | Brenner | ($19.95) | $18 | * |
| "Assembly Language for Applesoft Programmers" | Finley & Myers | ($18.95) | $18 | * |
| "Now That You Know Apple Assembly Language" | Jules Gilder | ($19.95) | $18 | * |
| "Enhancing Your Apple II, vol. 1" | Don Lancaster | ($15.95) | $15 | * |
| "Enhancing Your Apple II, vol. 2" | Don Lancaster | ($17.95) | $17 | * |
| "Assembly Cookbook for the Apple II/IIe" | Don Lancaster | ($21.95) | $20 | * |
| "Microcomputer Graphics" | Roy E.Myers | ($14.95) | $14 | * |
| "Assembly Lines -- the Book" | Roger Wagner | ($19.95) | $12 | * |

* These items add $2 for first item, $.75 for each additional item for US shipping.
+ Inquire for shipping cost.
Customers outside USA inquire for postage needed.
Texas residents please add 8% sales tax to all orders.
<< Master Card, VISA, Discover and American Express >>

**S-C Software Corporation**
2331 Gus Thomasson #125
DALLAS, TX 75228
Phone 214-324-2050

contains a command code, and the next two bytes point to a parameter block. ProDOS-16 uses two bytes for the command code and four bytes for the parameter block address.

Robert Lissner uses a simple system of vectoring all subroutine calls through several JMP vectors throughout AppleWorks. Some of his subroutines pass all their data in the three 6502 registers, some use fixed locations in page zero or in the $800-$FFF area, and some use a standard calling sequence with parameters after the JSR. One set of JMP vectors starts at $1000, and is used for calling all of the APLWORKS.SYSTEM subroutines. Another set begins at $D002 in the alternate $D000 bank of RAM, where either SEG.00 or SEG.XM has been loaded. Each overlay segment also begins with a JMP vector.

I have shown the JMP vector for APLWORKS.SYSTEM in lines 1520-2030 of the listing. To save space in the newsletter, I plugged in actual hexadecimal addresses for all those subroutines which are not listed in this issue. Where I have given them names, I included them as comments. The rest of them I will name later, when I get to them and figure out what they do and think of a unique meaningful moniker.

When AppleWorks is first fired up by executing APLWORKS.SYSTEM, one of the tasks is to look for either a 64K (or larger) memory card in the auxiliary slot or a card like RamFactor in one of the other slots. If it finds a RamFactor-like card with enough free memory, it loads SEG.XM into the 4K area at $D000. (RamFactor is Applied Engineerings version of the Apple Memory Card, and of course there are other companies also making these kinds of cards. In the rest of this article I will call this kind of memory SlotRAM memory.) If there is not enough SlotRAM memory available but there is 64K in the AuxRAM, AppleWorks loads SEG.00 instead.

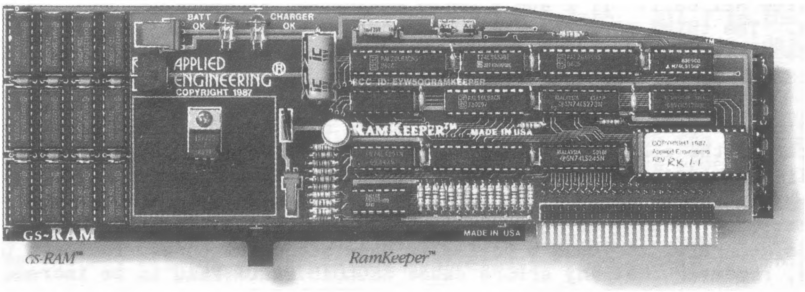
How can it do that? ProDOS supposedly has that $D000 space all tied up!

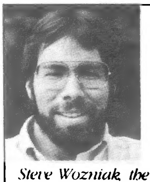
Well, ProDOS claims it all, but only really USES from $D100 through $D3FF. This is where the standard QUIT code is kept. During initialization AppleWorks copies that $300-byte area to the SEG.PR file, and then loads the appropriate SEG.00 or SEG.XM file. When you QUIT out of AppleWorks, it copies those $300 bytes from SEG.PR back into $D100 before doing the ProDOS Quit call. Note: AppleWorks only saves and restores $300 bytes! If you are using a non-standard Quit processor which takes over $300 bytes, running AppleWorks will clobber it. You will then have to reboot after quitting AppleWorks.

There are 24 subroutines inside the $D000 area which are accessed through a JMP vector starting at $D002. Depending on which SEG.xx is loaded there, they either talk to AuxRAM or SlotRAM. The routines that are of interest this month are equated in lines 1200-1220; listing them will have to wait for a future issue.

I defined some useful macros in lines 1320-1510. Macros are gradually growing on me. When I first added them to the S-C Assembler II, creating the S-C Macro Assembler version 1.0, I really couldn't think of many uses for them beyond sales appeal. Then I started using them for generating various types of data lists, and often-used code sequences like MLI calls. Now I am finding more and more uses.

The MLI.SL macro is a slight modification of my standard MLI-call macro. I added lines 1360-1400 to generate the error-tracking code which Lissner


BATT OK
CHARGER OK
APPLIED ENGINEERING
COPYRIGHT 1987
RAMKEEPER™ MADE IN USA
GS-RAM
MADE IN USA
GS-RAM™
RamKeeper™



uses. After nearly every call to MLI he uses a BEQ to branch around an INC of the error flag. ProDOS returns with status EQ and carry clear if there was no error, or NE and carry set if there was an error. The various ProDOS manuals make it clear that the carry status is supposed to be the preferred error flag, and I always got the impression that future versions might not support the EQ/NE method. Well, now they will HAVE to continue that support, because the world's most popular Apple program says so. Most other software I have looked at or written uses the CC/CS method, including such basic software as BASIC.SYSTEM.

If I use the SLI.ML macro with only two parameters, it generates only the two-line MLI call. If I add a third parameter, it generates the two extra lines. The third parameter in this macro is never actually used, just counted. The .DO ]#>2 line says "only assemble the following lines (up to the .FIN line) if the number of parameters is greater than two."

By using the private label :1 in the macro definition I avoid having to clutter up the code with lots of extra local or normal labels. This makes the listing easier to read with understanding. For example, look at lines 4340-4360. Those three lines actually generate 12 lines of code with three labels. You do have to remember when you are reading the code what these macros generate, if you are trying to understand the code. In these three lines, remember that any errors cause SEGLOAD.ERROR.FLAG to be incremented. If you want to, you can leave out the .LIST MOFF line that I used at the beginning (the unlisted line 1010). Then all of the code generated by each macro will be listed during assembly. I wanted to suppress the macro expansion listing to make the code easier to understand and to make it fit in the newsletter.

The HS macro is entirely for the purpose of shortening the listing. I use it in the SEGMENT.TABLE definition in lines 2330-2590. This table would take up two or three times as much paper if I did not use the macro.

The MSG macro is useful for defining strings that begin with a character count and include only printing ASCII characters. Since almost all of the text messages included in AppleWorks are like that, MSG is quite useful. I used it here in line 5230.

The SEGMENT.TABLE in lines 2230-2600 keeps track of vital information about the segments. There are four bytes for each segment. The first byte is the page number the segment should be loaded at. For example, Segment 01 has 35 in this byte, so it should be loaded at $3500.

The second byte of each group of four is used to keep track of how long it has been since the segment was loaded. Each time a segment is loaded the second byte for its entry in the SEGMENT.TABLE is zeroed while the second byte for all other entries is incremented, by the code in lines 3220-3360. I call this byte the "age" of a segment. I have not yet found any code which takes advantage of the information in the age-byte, but at least it is there. It may be, or could be, used to determine which segments to keep in AuxRAM or SlotRAM if there is not room for all of them.

The third and fourth bytes of each four-byte entry are 0000 if the segment must be loaded from disk. Naturally, this is what they are initially. After a segment is loaded from disk, if there is room in AuxRAM or SlotRAM it is also copied there. Then these two bytes in the segment table are set to a coded address so that the segment can be downloaded from RAM the next time it is needed.

When you select one of the three applications from the main menu in AppleWorks, LOAD.PROGRAM.SEGMENT.A will be called. I put a lot of information about the calling sequence of this program in the comments in lines 3000-3140. Basically, the segment number will be in the A-register. Lines 3170-3210 save this number and multiply it by four to make an index into the segment table. As already mentioned, lines 3220-3360 then increment the second byte of all entries and zero the second byte for the entry for te segment we want to load.

Lines 3370-3430 check to see if this segment is the same as the last one loaded. If it is, there is nothing left to do so it exits. I say "exits" rather than "returns" because it may or may not return. Bit 7 in the A-register at the time of call controls whether it returns after loading a segment or jumps into the loaded segment. If it does the latter, it jumps to $xx02, where xx is the page number the segment was loaded into. In this case, the X-register is a function number to be interpreted by the segment. When the segment is finished, it may return with an RTS to the program which called LOAD.PROGRAM.SEGMENT.A. The exit options are processed in lines 4810-4880.

If the segment is not the same as the most recently loaded one, line 3430 stores the new segment number in the CURRENTLY.LOADED.SEGMENT variable.

Lines 3440-3520 pick up the load address byte out of the segment table and install it in the various places it will be needed later. If that byte is 00, then the segment does not exist and the program returns with an RTS. This should never happen, of course, and I presume if it did it would be a bug. There are three null segments (0F, 14, and 15), but I would be surprised if any useful purpose is served by trying to load them. On the other hand, if there is some code somewhere which attempts to load all the segments in a range so that they will be copied in to AuxRAM or SlotRAM memory, the null segments might be included in the range without any disastrous effects.

Lines 3530-3570 treat segment $20 in a special way. I am not sure what that segment is, or why it is special. If you are trying to load segment $20 and the flag at $EA7 is non-zero it will not be loaded. Instead the loader will exit, either with an RTS or by using the function call (JMP to $4502 with a code in the X-register). I presume that $EA7 is set non-zero when function $20 is loaded, and stays that way until it is covered up by something else. This would let functions within segment $20 be called without reloading it unnecessarily even when other segments have been loaded after it was. I don't know, it sounds sort of kludgy. I'll just have to wait until I have looked into and disassembled a lot more of the AppleWorks code.

Lines 3580-3690 make sure that the variable CURRENT.APPLICATION is changed whenever you load segments $01, $10, or $18. I haven't found any use for this yet, and I am just hoping I have correctly guessed its significance. I have assumed that those segment numbers are the initially segments for each of the three applications, and have so indicated in the comments in lines 2130-2210.

Lines 3700-3750 look at the third and fourth bytes in the segment table entry to see if they are 0000. If so, the segment must be loaded from the AppleWorks Program disk. If not, lines 3760-3810 download the segment from AuxRAM or SlotRAM memory. The downloading is handled by a subroutine from either SEG.00 or SEG.XM, which I will probably describe in detail in a future issue of AAL. They handle AuxRAM or SlotRAM in interesting ways.

If a segment must be loaded from disk, the subroutine beginning at line 3830 takes over. Lines 3840-3920 modify the general SEG.xx pathname buffer to point to either SEG.M0 or SEG.M1. For some reason the first nine segments are stored on SEG.M0 and the rest on SEG.M1. Once upon a time it probably made sense....

Lines 3930-3970 will then try to open the selected SEG.Mx file. If the file will not open, AppleWorks assumes the only possible reason could be that the disk is not mounted and calls CALL.FOR.AWPROGRAM.DISK (lines 2780-2950) to tell you to do so. The message says "Place the AppleWorks PROGRAM disk in Drive 1 and press Return." Actually you SHOULD be able to place the disk in ANY drive, if you have specified a complete pathname for the program disk. And, actually, you can press any key, not just RETURN.

Once the file has been opened successfully, lines 3990-4050 store the reference number ProDOS assigns the file in all the other IOBs. Line 4060 calls CLR.PRODOS.BITMAP to make sure ProDOS will allow reading into the range $800-9FFF. Lines 4070-4080 clear a byte used as an error flag for all the subsequent MLI calls. After going through all the motions of loading the segment this flag will still contain a zero if no errors were reported by ProDOS.

Lines 4090-4120 read in the first 140 bytes of the SEG.Mx file. The front of the file contains a segment offset table with 3-byte values for each segment. These three bytes tell what offset (or MARK) value to send via the MLI "Set Mark" call before reading in the segment. Subtracting a segment's offset from that of the next segment gives the length of the segment we want to load. Since there are 43 segments, will need 43 3-byte values for starting addresses, plus one more for the zero-th entry, plus still one more for computing the length of the 43rd segment. That is a total of 45*3 bytes, or 135. Appleworks reads 140, which allows for one more segment to be added without changing this constant.

Note that the program goes right on reading the SEG.Mx file whether there is a reading error or not. SEGLOAD.ERROR.FLAG gets incremented if there is an error, but nothing else is done about it until we have gone through all the motions of loading the segment and closing the file. If there was any error at all, lines 4380-4390 find it out and return with an RTS to whoever called the segment loader. This seems a little dangerous, because the user will never know what happened. A glitched AppleWorks Program disk could cause very erratic behavior without any explanation, yet AppleWorks COULD have reported what was wrong and exactly which segment could not be loaded.

Lines 4130-4180 multiply the segment number by 3 to get a pointer into the SEG.Mx segment offset table. Lines 4190-4320 pick up the offset and save it for the Set Mark MLI call, and also compute the segment length for the Read MLI call. Lines 4340-4360 set the mark, read the segment, and close the file. After the segment has been read, line 4370 calls SET.PRODOS.BITMAP to re-protect all RAM from $800 through $9FFF.

The first two bytes of every segment on both SEG.Mx files contains the end address plus 1 of that segment. This value may be different from the number of bytes loaded plus the starting address, if the segment needs some private RAM at the end of itself. However, I haven't found any cases where this is so. We already knew the length in order to read the segment from

## SPECIAL !!! EXPANDED RAM/ROM BOARD: $39.00

Similar to our $30 RAM/ROM dev board described below. Except this board has two sockets to hold your choice of 2-2K RAM, 2-2K ROM or even 2-4K ROM for a total of 8K. Mix RAM and ROM too. Although Apple limits access to only 2K at a time, soft switches provide convenient socket selection. Hard switches control defaults.

## IMPROVED ] [ [ ][ IN A MAC (ver 2.0): $75.00

Now includes faster graphics, UniDisk support and more! Bi-directional data transfers are a snap! This Apple II emulator runs DOS 3.3/PRODOS (including 6502 machine language routines) on a 512K MAC or MACPLUS. All Apple II features are supported such as HI/LO-RES graphics, 40/80 column text, language card and joystick. Also included: clock, RAM disk, keyboard buffer, on-screen HELP, access to the desk accessories and support for 4 logical disk drives. Includes 2 MAC diskettes (with emulation, communications and utility software, plus DOS 3.3 and PRODOS system masters, including Applesoft and Integer BASIC) and 1 Apple II diskette.

## SCREEN.GEN: $35.00

Develop HI-RES screens for the Apple II on a Macintosh. Use MACPAINT (or any other application) on the MAC to create your Apple II screen. Then use SCREEN.GEN to transfer directly from the MAC to an Apple II (with SuperSerial card ) or IIc. Includes Apple II diskette with transfer software plus fully commented SOURCE code.

## MIDI-MAGIC for Apple //c: $49.00

Compatible with any MIDI equipped music keyboard, synthesizer, organ or piano. Package includes a MIDI-out cable (plugs directly into modem port - no modifications required!) and 6-song demo diskette. Large selection of digitized QRS player-piano music available for 19.00 per diskette (write for catalog). MIDI-MAGIC compatible with Apple II family using Passport MIDI card (or our own input/output card w/drum sync for only $99.00).

## FONT DOWNLOADER & EDITOR: $39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. Special functions (like expanded, compressed etc.) supported. Includes HIRES screen editor to create custom fonts and special graphics symbols. For Apple II, II+, //e. Specify printer: Apple Imagewriter,Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/85, or Okidata 92/192

*** FONT LIBRARY DISKETTE #1: $19.00** contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

## DISASM 2.2e $30.00 ($50.00 with SOURCE Code)

Use this intelligent disassembler to investigate the inner workings of Apple II machine language programs. DISASM converts machine code into meaningful, symbolic source compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even provides label substitution. Address-based triple cross reference generator included. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his ASSEMBLY COOKBOOK.

## The 'PERFORMER' CARD: $39.00 ($59.00 with SOURCE Code)

Converts a 'dumb' parallel printer I/F card into a 'smart' one. Simple command menu. Features include perforation skip, auto page numbering with date & title, large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93.

## 'MIRROR' ROM: $25.00 ($45.00 with SOURCE Code)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support.

## RAM/ROM DEVELOPMENT BOARD: $30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into $Cn00-CnFF and $C800-CFFF.

## C-PRINT For The APPLE //c: $69.00

Connect standard parallel printers to an Apple //c serial port. Separate P/S included. Just plug in and print!

---

Unless otherwise specified, all Apple II diskettes are standard (not copy protected!) 3.3 DOS.

Avoid a $3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders OK.

RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885 disk, so recomputing the same value seems like make-work. But who knows? Lines 4400-4520 pick up the address from the beginning of the segment, subtract the loading address, and store the result in SL.LEN. That gives the "uploader" the number of bytes to copy into AuxRAM or SlotRAM.

Lines 4530-4610 compute the actual address of the current segment's entry in the segment table, and save this address at SL.SEG. We are gradually building up the calling sequence for the "uploader". Lines 4620-4710 determine whether there is enough RAM left in either AuxRAM or SlotRAM, whichever is being used, for uploading the segment. If so, lines 4720-4770 call on the uploader to do so. The uploader will set the 3rd and 4th bytes for the segment in the segment table so that future calls to load this segment can download them from RAM instead of reading the AppleWorks Program disk.

Lines 4810-4880 have already been discussed above. They decide whether to return to the caller with an RTS, or to JMP into the segment just loaded with a function code in the X-register. Lines 4930-5200 are the parameter blocks, or IOBs, used by the various MLI calls in the segment loader.

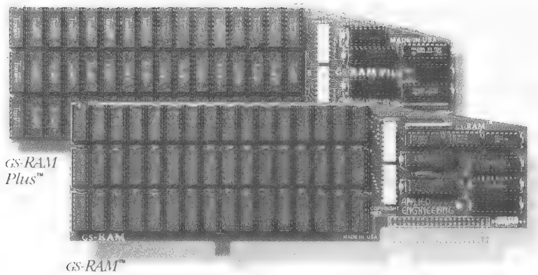
Lines 5220-5240 define the message used by the program which prompts you to mount the AppleWorks program disk. I used the .PH and .EP because this message does not immediately follow the IOBs in the real code. The .PH address shows where it really is assembled in version 1.3. Lines 5250-5400 are the two subroutines for de-protecting and protecting RAM from $800 through $9FFF.

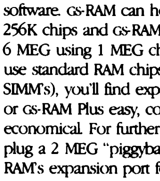
## Assembly Language Programmers Wanted

The American Printing House for the Blind (APH) is looking for assembly language programmers to write and modify educational and applications software for the Apple II series. People interested should be familiar with 6502, 65C02, and 65C816 processors. They should also know both DOS 3.3 and ProDOS. Knowledge of the Z-80 is also highly desirable. The position requires some travel, speaking, and writing. Interested candidates should send a resume and some example code to:

American Printing House for the Blind
Larry Skutchan
P. O. Box 6085
Louisville, KY 40206

An Equal Opportunity Employer

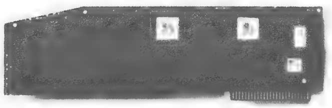




```
               1000 *SAVE AW.SRC.V8N6

               1020 ***missing line above was ".LIST MOFF", which
               1030 ***shrinks the listing by not listing macro expansions
               1040 *---AppleWorks Variables---------
0900-          1050 BUF.900                  .EQ $0900
0A00-          1060 STR.A00                  .EQ $0A00
0EA7-          1070 X.0EA7                   .EQ $0EA7
0F14-          1080 SEGMENT.ADDRESS          .EQ $0F14
0FCE-          1090 X.0FCE                   .EQ $0FCE
0FF3-          1100 X.0FF3                   .EQ $0FF3
0FF5-          1110 FREE.MEMORY.xxxxK        .EQ $0FF5
               1120 *--------------------------------
1FF5-          1130 DISPLAY.ON.LINE.23       .EQ $1FF5
1D35-          1140 AW.KEYIN                 .EQ $1D35
B700-          1150 X.B700                   .EQ $B700
               1160 *---ProDOS Global Page-----------
BF00-          1170 MLI                      .EQ $BF00
BF58-          1180 PRODOS.BITMAP            .EQ $BF58
               1190 *---Subroutines from SEG.00 or SEG.XM---
D002-          1200 Measure.free.Memory                  .EQ $D002
D005-          1210 Download.from.AuxRAM.or.Memory.Card .EQ $D005
D011-          1220 Upload.to.AuxRam.or.Memory.Card     .EQ $D011
               1230 *---Page Zero Variables----------
85-            1240 Z.85                     .EQ $85
8D-            1250 SEGLOAD.ERROR.FLAG       .EQ $8D
9A-            1260 P0                       .EQ $9A
9E-            1270 P4                       .EQ $9E
9F-            1280 P5                       .EQ $9F
A4-            1290 DISPLAY.ACTIVE.FLAG      .EQ $A4
               1300 *--------------------------------
               1310        .PH $1000
               1320 *--------------------------------
               1330        .MA MLI.SL   Macro for MLI calls
               1340        JSR MLI
               1350        .DA #$]1,SLIOB.]2
               1360     .DO ]#>2        If 3rd parameter, include these lines
               1370        BEQ :1       ...no error reported by MLI
               1380        INC SEGLOAD.ERROR.FLAG   error, so set flag
               1390 :1
               1400     .FIN
               1410        .EM
               1420 *--------------------------------
               1430        .MA HS       Macro to shrink listing only
               1440        .HS ]1
               1450        .EM
               1460 *--------------------------------
               1470        .MA MSG      Macro to make a string with first byte
               1480        .DA #:1-*-1  giving the length, rest is ASCII.
               1490        .AS "]1"
               1500 :1
               1510        .EM
               1520 *--------------------------------
1000- 4C 25 3E 1530    JMP $3E25 RELOCATE.AND.START+$1000
1003- 4C 86 11 1540        JMP CALL.FCR.AWPROGRAM.DISK
1006- 4C A1 11 1550    JMP $11A1 LOAD.PROGRAM.SEGMENT.A
1009- 4C 41 13 1560    JMP $1341 APPEND.STRINGS
100C- 4C 66 13 1570    JMP $1366 CLEAR.MAIN.WINDOW
100F- 4C 6E 13 1580    JMP $136E
1012- 4C 9D 13 1590    JMP $139D CLR.LINE.X.TO.LINE.Y
1015- 4C D1 14 1600    JMP $14D1 DISPLAY.STRING   (A) bytes, starting at $80,81
1018- 4C 9D 17 1610    JMP $179D CONVERT.A.TO.RJBF.STRING
101B- 4C 15 18 1620    JMP $1815 HANG
101E- 4C D1 17 1630    JMP $17D1 DIVIDE.P0.BY.P2
1021- 4C 18 18 1640    JMP $1818 BEEP.AND.CLEAR.KEYBUF
1024- 4C 23 18 1650    JMP $1823 MOVE.CURSOR.TO.XY
1027- 4C 37 18 1660    JMP $1837
102A- 4C 42 18 1670    JMP $1842
102D- 4C 50 18 1680    JMP $1850
1030- 4C 6C 18 1690    JMP $186C
1033- 4C 72 18 1700    JMP $1872
1036- 4C 7A 18 1710    JMP $187A COPY.SCRN.LINE.TO.0900
1039- 4C B4 18 1720    JMP $18B4 GET.A.PARMS
103C- 4C 1D 19 1730    JMP $191D
103F- 4C 77 1A 1740    JMP $1A77
1042- 4C FC 1A 1750        JMP SET.PRODOS.BITMAP
1045- 4C 00 1B 1760        JMP CLR.PRODOS.BITMAP
1048- 4C 0B 1B 1770    JMP $1B0B DRAW.TOP.AND.BOTTOM.LINES
104B- 4C 2B 1B 1780    JMP $1B2B
104E- 4C 3A 1B 1790    JMP $1B3A MULTIPLY.X.BY.Y
1051- 4C 4E 1B 1800    JMP $1B4E MULTIPLY.P0.BY.P2
1054- 4C 84 1B 1810    JMP $1B84 MOVE.BLOCK.DOWN
1057- 4C AC 1B 1820    JMP $1BAC MOVE.BLOCK.UP
105A- 4C DF 1B 1830    JMP $1BDF
105D- 4C F1 1B 1840    JMP $1BF1 WAIT.FOR.SPACE.RETURN.OR.ESCAPE
1060- 4C 21 1C 1850    JMP $1C21 PRINTER.DRIVER
1063- 4C 0F 1D 1860    JMP $1D0F
1066- 4C 35 1D 1870    JMP $1D35 AW.KEYIN
1069- 4C 80 1E 1880    JMP $1E80 MOVE.CURSOR.TO.TCOL.TROW
```

```
106C- 4C 8A 1E 1890    JMP $1E8A
106F- 4C B4 1E 1900    JMP $1EB4 MAP.LOWER.TO.UPPER   char in A-reg
1072- 4C BF 1E 1910    JMP $1EBF FILTER.LC.TO.UC   string
1075- 4C D9 1E 1920    JMP $1ED9 COMPARE.STRINGS
1078- 4C F8 1E 1930    JMP $1EF8 MOVE.STRING
107B- 4C 3E 1F 1940    JMP $1F3E DISPLAY.AT
107E- 4C 29 20 1950    JMP $2029
1081- 4C D1 1F 1960    JMP $1FD1 DELAY.A.TENTHS
1084- 4C E0 1F 1970    JMP $1FE0 CLEAR.KEYBUF
1087- 4C 93 20 1980    JMP $2093 DISPLAY.STRING.PO
108A- 4C E9 1F 1990    JMP $1FE9 DISPLAY.TOKEN.X
108D- 4C AE 20 2000    JMP $20AE
1090- 4C BE 20 2010    JMP $20BE
1093- 4C F5 1F 2020    JMP $1FF5 DISPLAY.ON.LINE.23
1096- 4C D6 20 2030    JMP $20D6
               2040 *-------------------------------
               2050 *   CONSTANTS & VARIABLES
               2060 *-------------------------------
1099- 00 0A    2070 HANDLE.STR.A00       .DA STR.A00
109B- A7 10    2080 HANDLE.SEGMENT.TABLE .DA SEGMENT.TABLE
               2090 *-------------------------------
               2100 *   Holds result after calling CONVERT.A.TO.RJBF.STRING
               2110 *   but this result is apparently never referenced.
109D-          2120 RJBF.STRING >HS 03.20.20.20
               2130 *-------------------------------
               2140 *   10A1 holds 00, 01, 02, or 03, indicating which of segments
               2150 *       1, 16, or 24 was the most recently loaded.
               2160 *       00 means none of these have yet been loaded.
               2170 *       01 means segment 1  (Data Base)
               2180 *       02 means segment 16 (Word Processor)
               2190 *       03 means segment 24 (Spread Sheet)
10A1- 00       2200 CURRENT.APPLICATION     .HS 00
10A2-          2210 APPLICATION.SEGMENT.LIST >HS FF.01.10.18
               2220 *-------------------------------
               2230 *   Segment Table
               2240 *       There are 43 program segments in files SEG.M0 and SEG.M1
               2250 *       Four bytes in this table for each segment.
               2260 *          Byte 1:  Starting page to load segment into.
               2270 *          Byte 2:  Age of segment (00=just loaded, FF=oldest)
               2280 *       Bytes 3,4:  0000 if must be loaded from disk
               2290 *                   xxxx if in RAM or Aux RAM
               2300 *-------------------------------
10A6- FF       2310 CURRENTLY.LOADED.SEGMENT .HS FF
               2320 SEGMENT.TABLE
10A7-          2330        >HS 30.000000            Dummy Entry, seg 00
10AB-          2340        >HS 35.000000            Seg 01 (Data Base)
10AF-          2350        >HS 45.000000.45.000000  Segs 02,03
10B7-          2360        >HS 45.000000.45.000000  Segs 04,05
10BF-          2370        >HS 45.000000.4E.000000  Segs 06,07
10C7-          2380        >HS 4E.000000.4A.000000  Segs 08,09
10CF-          2390        >HS 4E.000000.4E.000000  Segs 0A,0B
10D7-          2400        >HS 4E.000000.53.000000  Segs 0C,0D
10DF-          2410        >HS 4E.000000.00.000000  Segs 0E,0F
               2420 *
10E7-          2430        >HS 35.000000            Seg 10 (Word Processor)
10EB-          2440        >HS 3D.000000.3D.000000  Segs 11,12
10F3-          2450        >HS 40.000000.00.000000  Segs 13,14
10FB-          2460        >HS 00.000000.67.000000  Segs 15,16
1103-          2470        >HS 77.000000            Seg 17
               2480 *
1107-          2490        >HS 33.000000            Seg 18 (Spread Sheet)
110B-          2500        >HS 3B.000000.53.000000  Segs 19,1A
1113-          2510        >HS 53.000000.53.000000  Segs 1B,1C
111B-          2520        >HS 53.000000.53.000000  Segs 1D,1E
1123-          2530        >HS 53.000000.45.000000  Segs 1F,20
112B-          2540        >HS 64.000000.64.000000  Segs 21,22
1133-          2550        >HS 64.000000.64.000000  Segs 23,24
113B-          2560        >HS 64.000000.64.000000  Segs 25,26
1143-          2570        >HS 64.000000.64.000000  Segs 27,28
114B-          2580        >HS 64.000000.64.000000  Segs 29,2A
1153-          2590        >HS 64.000000            Seg 2B
B0-            2600 SEGMENT.TABLE.SIZE .EQ *-SEGMENT.TABLE
               2610 *-------------------------------
1157- 00       2620 POSITION.IN.STRING .HS 00   Used by DISPLAY.STRING
               2630 *-------------------------------
               2640 *   Note **SECRET** limitation:  the pathname
               2650 *       to the SEG.M0 and SEG.M1 files is limited
               2660 *       to a total of 29 bytes, including the "/".
1158- 01 2F    2670 SEGMENT.PATHNAME .HS 01.2F
115A-          2680                  .BS 28
               2690 *-------------------------------
1176- 01       2700 KEYIN.CURSOR.TYPE .HS 01 00=underline, 01=flashing
1177- 01       2710 KEYIN.CURSOR.FLAG .HS 01 00=no cursor, 01=cursor
1178- 00       2720 SCROLL.DIRECTION  .HS 00 Used by DISPLAY.STRING
1179- 00       2730 BYTES.IN.STRING   .HS 00 Used by DISPLAY.STRING
               2740 *-------------------------------
117A-          2750 KEYBUF     .BS 10
1184- 00       2760 KEYBUF.IN  .HS 00
1185- 00       2770 KEYBUF.OUT .HS 00
```



*PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.*

### A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

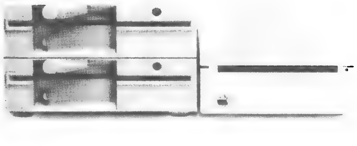
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5"disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIGS, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

*PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.*



### Versatile data storage.

PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data,but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS storage and a 143K Apple 5.25" drive for MS-DOS storage.

### Make your Apple speak IBM.

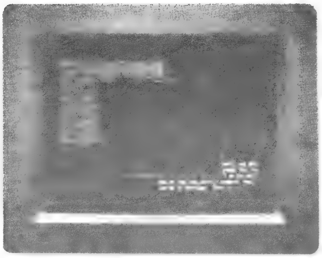
PC Transporter memory choices.

| RAM in Apple mode: | RAM in IBM mode: | Price: |
|---|---|---|
| 384K | 256K | $489.00 |
| 512K | 384K | 529.00 |
| 640K | 512K | 569.00 |
| 768K | 640K | 609.00 |

Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

| | |
|---|---|
| **IIGS Installation Kit** | 49.00 |
| **IIe/II Plus Installation Kit** | 39.00 |
| **PC Transporter Accessories** | |
| **5.25" IBM Format 360K Drive Systems** | |
| **Single-Drive System** | 269.00 |
| **Dual-Drive System** | 399.00 |
| **Half-Height Drive** (add to Single-Drive system to make Dual-Drive) | 135.00 |
| **IBM-Style Keyboard** (required for Apple II Plus. Requires IBM Keyboard Cable.) | 139.00 |
| **IBM Keyboard Cable** | 34.00 |
| **Sony RGB Monitor** | 499.00 |
| **Analog RGB Cable** (for use with Sony monitor) | 39.00 |
| **Digital RGB Cable** (for use with Sony monitor) | 39.00 |
| **Digital RGB Adapter** | 24.00 |
| **ColorSwitch** (included with IIGS Installation Kit) | 44.00 |
| **128K ZIP PC Transporter Memory Expansion Chip Set** | 40.00 per set |
| **8087-2 Math Co-processor Chip** | 229.00 |
| **Heavy Duty Power Supply** (IIe and II Plus only) | 69.00 |

See your dealer or call or send check or money order to Applied Engineering. MasterCard, VISA and COD welcome. Texas residents add 6¼% sales tax.



*PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software!*

*And, you can also use an Apple composite monitor in IBM text or graphics mode.*

### Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

### Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

### How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

**AE Applied Engineering**
***The Apple enhancement experts.***
P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation.

Apple II Plus must be FCC Certified. IBM and PC/XT are registered trademarks of International Business Machines. Lotus and 1-2-3 are registered trademarks of Lotus Development Corporation. MultiMate and dBASE III PLUS are registered trademarks of Ashton-Tate, Inc. MS and Flight Simulator are registered trademarks of Microsoft. Apple IIe and ProDOS are registered trademarks and IIGS and UniDisk are trademarks of Apple Computer.

```
                  2780 *--------------------------------
                  2790 *   Subroutine to request mounting of the AppleWorks
                  2800 *       Program Disk, so a SEG.xx file can be opened.
                  2810 *--------------------------------
                  2820 * (1186) 1003 124B 275F
                  2830 CALL.FOR.AWPROGRAM.DISK
1186- A5 A4       2840        LDA DISPLAY.ACTIVE.FLAG     Save Display lockout flag
1188- 48          2850        PHA
1189- A9 00       2860        LDA #0       Allow display
118B- 85 A4       2870        STA DISPLAY.ACTIVE.FLAG
118D- 20 F5 1F    2880        JSR DISPLAY.ON.LINE.23   "Place the AppleWorks
1190- C1 13       2890        .DA MSG..1   PROGRAM disk in Drive 1 and press Return."
1192- 20 35 1D    2900        JSR AW.KEYIN       Wait for Any Key
1195- A9 00       2910        LDA #0       Indicate Program Disk mounted
1197- 8D CE 0F    2920        STA X.0FCE
119A- 68          2930        PLA                 Restore Display lockout flag
119B- 85 A4       2940        STA DISPLAY.ACTIVE.FLAG
119D- 60          2950        RTS
                  2960 *--------------------------------
119E-             2970 SEGMENT.INDEX  .BS 1
119F-             2980 SEGMENT.SAVEX  .BS 1
11A0-             2990 SEGMENT.NUMBER .BS 1
                  3000 *--------------------------------
                  3010 *   (A)=Segment Number or Segment Number + $80.
                  3020 *       There are 43 segments, numbered 1 to 43.
                  3030 *       Segments 1-9 are in file SEG.M0, and
                  3040 *       segments 10-43 are in file SEG.M1.
                  3050 *
                  3060 *       The Segment is loaded at $xx00, where xx
                  3070 *       is the first byte for the entry in the
                  3080 *       Segment Table.
                  3090 *
                  3100 *       Bit 7 of A controls the execution option.
                  3110 *          If 0, then exit with JMP $xx02.
                  3120 *          If 1, then exit with an RTS.
                  3130 *   (X)=Function Number in segment
                  3140 *--------------------------------
                  3150 * (11A1) 1006 1877
                  3160 LOAD.PROGRAM.SEGMENT.A
11A1- 8D A0 11    3170        STA SEGMENT.NUMBER
11A4- 8E 9F 11    3180        STX SEGMENT.SAVEX
11A7- 0A          3190        ASL          *4 to get index into Segment Table
11A8- 0A          3200        ASL
11A9- 8D 9E 11    3210        STA SEGMENT.INDEX
                  3220 *---Increment all entries--------
11AC- A2 B0       3230        LDX #SEGMENT.TABLE.SIZE
11AE- CA          3240 .1     DEX          For X = $AC to $04 step -4
11AF- CA          3250        DEX
11B0- CA          3260        DEX
11B1- CA          3270        DEX
11B2- FE A8 10    3280        INC SEGMENT.TABLE+1,X
11B5- D0 03       3290        BNE .2
11B7- DE A8 10    3300        DEC SEGMENT.TABLE+1,X      max value is $FF
11BA- E0 00       3310 .2     CPX #0
11BC- D0 F0       3320        BNE .1
                  3330 *---Clear indexed entry----------
11BE- AE 9E 11    3340        LDX SEGMENT.INDEX
11C1- A9 00       3350        LDA #0
11C3- 9D A8 10    3360        STA SEGMENT.TABLE+1,X
                  3370 *---Keep track of loaded segment--
11C6- AD A0 11    3380        LDA SEGMENT.NUMBER
11C9- 29 7F       3390        AND #$7F
11CB- CD A6 10    3400        CMP CURRENTLY.LOADED.SEGMENT
11CE- D0 03       3410        BNE .3
11D0- 4C 18 13    3420        JMP LOAD.EXIT.BY.OPTION  Already loaded, so we're done!
11D3- 8D A6 10    3430 .3     STA CURRENTLY.LOADED.SEGMENT
                  3440 *---Get Load Address-------------
11D6- AE 9E 11    3450        LDX SEGMENT.INDEX
11D9- BC A7 10    3460        LDY SEGMENT.TABLE,X
11DC- D0 03       3470        BNE .4
11DE- 4C 23 13    3480        JMP LOAD.EXIT.RTS    ...no such segment, quit now
11E1- 8C 2D 13    3490 .4     STY SLIOB.READ.SEGMENT+3
11E4- 8C 29 12    3500        STY .10+1    In call to Downloader Subroutine
11E7- 8C 0E 13    3510        STY SL.ADR+1
11EA- 8C 22 13    3520        STY SEGMENT.EXECUTION.VECTOR+2
                  3530 *---Is this segment $20?---------
11ED- C9 20       3540        CMP #$20
11EF- D0 05       3550        BNE .5
11F1- AE A7 0E    3560        LDX X.0EA7
11F4- D0 34       3570        BNE .11        ...Exit
                  3580 *---Keep track of application----
11F6- AE A1 10    3590 .5     LDX CURRENT.APPLICATION
11F9- F0 05       3600        BEQ .6
11FB- DD A2 10    3610        CMP APPLICATION.SEGMENT.LIST,X
11FE- F0 2A       3620        BEQ .11        ...Exit
1200- A2 03       3630 .6     LDX #3
1202- DD A2 10    3640 .7     CMP APPLICATION.SEGMENT.LIST,X
1205- F0 05       3650        BEQ .8
1207- CA          3660        DEX
1208- D0 F8       3670        BNE .7
120A- F0 03       3680        BEQ .9         ...not in the list
120C- 8E A1 10    3690 .8     STX CURRENT.APPLICATION
```

```
               3700 *---Decide how to load it--------
120F- AE 9E 11 3710 .9     LDX SEGMENT.INDEX    If address in SEGMENT.TABLE is 0000,
1212- BD A9 10 3720        LDA SEGMENT.TABLE+2,X   then load from Program Disk;
1215- 8D 14 0F 3730        STA SEGMENT.ADDRESS     otherwise, download from Ram
1218- 1D AA 10 3740        ORA SEGMENT.TABLE+3,X
121B- F0 10    3750        BEQ LOAD.SEGMENT.FROM.DISK
121D- BD AA 10 3760        LDA SEGMENT.TABLE+3,X
1220- 8D 15 0F 3770        STA SEGMENT.ADDRESS+1
1223- 20 05 D0 3780        JSR Download.from.AuxRAM.or.Memory.Card
1226- 14 0F    3790        .DA SEGMENT.ADDRESS
1228- 00 00    3800 .10    .HS 00.00  Hi-byte filled in by program
122A- 4C 18 13 3810 .11    JMP LOAD.EXIT.BY.OPTION   ...Exit
               3820 *--------------------------------
               3830 LOAD.SEGMENT.FROM.DISK
122D- AC 58 11 3840        LDY SEGMENT.PATHNAME   Change name end to 'M0' or 'M1'
1230- A9 4D    3850        LDA #'M'
1232- 99 57 11 3860        STA SEGMENT.PATHNAME-1,Y
1235- A9 30    3870        LDA #'0'
1237- AE A6 10 3880        LDX CURRENTLY.LOADED.SEGMENT
123A- E0 0A    3890        CPX #10
123C- 90 02    3900        BCC .1        Segments 1-9 from SEG.M0
123E- A9 31    3910        LDA #'1'      Segments 10-43 from SEG.M1
1240- 99 58 11 3920 .1     STA SEGMENT.PATHNAME,Y
               3930 *---Attempt to open SEG.Mx-------
1243-          3940 .2    >MLI.SL C8,OPEN   Open the file
1249- F0 06    3950        BEQ .3        ...no problems
124B- 20 86 11 3960        JSR CALL.FOR.AWPROGRAM.DISK
124E- 4C 43 12 3970        JMP .2
               3980 *--------------------------------
1251- A9 00    3990 .3     LDA #0        Indicate Program Disk mounted
1253- 8D CE 0F 4000        STA X.0FCE
1256- AD 29 13 4010        LDA SLIOB.OPEN+5   Copy File RefNum to IOBs
1259- 8D 2B 13 4020        STA SLIOB.READ.SEGMENT+1
125C- 8D 33 13 4030        STA SLIOB.CLOSE+1
125F- 8D 35 13 4040        STA SLIOB.SETMARK+1
1262- 8D 3A 13 4050        STA SLIOB.READ.INDEX+1
1265- 20 00 1B 4060        JSR CLR.PRODOS.BITMAP
1268- A9 00    4070        LDA #0        Clear Error Flag
126A- 85 8D    4080        STA SEGLOAD.ERROR.FLAG
               4090 *---Read Segment Index-----------
               4100 *   $8C bytes at beginning of SEG.Mx file
               4110 *   into buffer at $0900.
126C-          4120       >MLI.SL CA,READ.INDEX,S
               4130 *---Form segnum*3 for index------
1276- AD A6 10 4140        LDA CURRENTLY.LOADED.SEGMENT
1279- 0A       4150        ASL
127A- 18       4160        CLC
127B- 6D A6 10 4170        ADC CURRENTLY.LOADED.SEGMENT
127E- AA       4180        TAX
               4190 *---Get Mark and Length----------
127F- A9 03    4200        LDA #3        Do for 3 loops
1281- 85 9A    4210        STA P0
1283- A0 00    4220        LDY #0        Start with lsb
1285- 38       4230        SEC
1286- BD 00 09 4240 .4     LDA BUF.900,X      Byte of Mark for this segment
1289- 99 36 13 4250        STA SLIOB.SETMARK+2,Y
128C- BD 03 09 4260        LDA BUF.900+3,X    Byte of Mark for next segment
128F- FD 00 09 4270        SBC BUF.900,X      Byte of Mark for this segment
1292- 99 2E 13 4280        STA SLIOB.READ.SEGMENT+4,Y   Byte of Length
1295- E8       4290        INX
1296- C8       4300        INY
1297- C6 9A    4310        DEC P0
1299- D0 EB    4320        BNE .4        More bytes
               4330 *---Read the Segment-------------
129B-          4340       >MLI.SL CE,SETMARK,S    Set Mark
12A5-          4350       >MLI.SL CA,READ.SEGMENT,S   Read Segment
12AF-          4360       >MLI.SL CC,CLOSE,S         Close File
12B9- 20 FC 1A 4370        JSR SET.PRODOS.BITMAP
12BC- A5 8D    4380        LDA SEGLOAD.ERROR.FLAG
12BE- D0 63    4390        BNE LOAD.EXIT.RTS   ...Error(s), give it up.
               4400 *---Save length of segment-------
12C0- A9 00    4410        LDA #0        Build pointer to segment in P4,P5
12C2- 85 9E    4420        STA P4
12C4- AD 22 13 4430        LDA SEGMENT.EXECUTION.VECTOR+2
12C7- 85 9F    4440        STA P5
12C9- A0 00    4450        LDY #0        Get first two bytes
12CB- B1 9E    4460        LDA (P4),Y    which are end address + 1
12CD- 8D 0F 13 4470        STA SL.LEN    and subtract load address
12D0- C8       4480        INY           to get length
12D1- B1 9E    4490        LDA (P4),Y    (I thought we already knew
12D3- 38       4500        SEC                the length!)
12D4- E5 9F    4510        SBC P5
12D6- 8D 10 13 4520        STA SL.LEN+1
```

```
                    4530 *---See if room to save segment in RAM---
12D9- AD 9E 11 4540        LDA SEGMENT.INDEX
12DC- 18       4550        CLC          Build pointer to vector in SEGMENT.TABLE
12DD- 69 02    4560        ADC #2
12DF- 6D 9B 10 4570        ADC HANDLE.SEGMENT.TABLE
12E2- 8D 0B 13 4580        STA SL.SEG
12E5- AD 9C 10 4590        LDA HANDLE.SEGMENT.TABLE+1
12E8- 69 00    4600        ADC #0
12EA- 8D 0C 13 4610        STA SL.SEG+1
12ED- 20 02 D0 4620        JSR Measure.free.Memory
12F0- AD F6 0F 4630        LDA FREE.MEMORY.xxxxK+1   If non-zero, at least 256K
12F3- D0 0E    4640        BNE .5                    ...which is plenty!
12F5- AD F5 0F 4650        LDA FREE.MEMORY.xxxxK
12F8- C9 10    4660        CMP #16                   Is there at least 16K?
12FA- B0 07    4670        BCS .5                    ...Yes, plenty
12FC- 0A       4680        ASL          Convert to # pages free
12FD- 0A       4690        ASL
12FE- CD 10 13 4700        CMP SL.LEN+1  Compare to what is needed
1301- 90 15    4710        BCC LOAD.EXIT.BY.OPTION   ...not enough room
1303- A9 01    4720 .5     LDA #1
1305- 8D F3 0F 4730        STA X.0FF3
1308- 20 11 D0 4740        JSR Upload.to.AuxRam.or.Memory.Card
130B- 00 00    4750 SL.SEG .HS 00.00    Address in SEGMENT.TABLE of vector
130D- 00 00    4760 SL.ADR .HS 00.00    Load Address of Segment
130F- 00 00    4770 SL.LEN .HS 00.00    Length of Segment
1311- A9 00    4780        LDA #0
1313- 8D F3 0F 4790        STA X.0FF3
1316- 85 85    4800        STA Z.85     ...fall into LOAD.EXIT...
               4810 *--------------------------------
               4820 LOAD.EXIT.BY.OPTION
1318- AE 9F 11 4830        LDX SEGMENT.SAVEX
131B- AD A0 11 4840        LDA SEGMENT.NUMBER
131E- 30 03    4850        BMI LOAD.EXIT.RTS
               4860 SEGMENT.EXECUTION.VECTOR
1320- 4C 02 FF 4870        JMP $FF02    Page value filled in by program
               4880 *                   so JMPs to $xx02 in segment.
               4890 *--------------------------------
1323- 60       4900 LOAD.EXIT.RTS RTS
               4910 *
               4920 *---OPEN IOB---------------------
               4930 SLIOB.OPEN
1324- 03 58 11
1327- 00 B7    4940        .DA #3,SEGMENT.PATHNAME,X.B700
1329- 00       4950        .HS 00       Open RefNum
               4960 *
               4970 *---READ IOB---------------------
               4980 SLIOB.READ.SEGMENT
132A- 04       4990        .DA #4
132B- 00       5000        .HS 00       RefNum
132C- 00 00    5010        .DA $0000    Load Address
132E- 00 38    5020        .DA $3800    Load Length
1330- 00 00    5030        .DA $0000    Actual Length
               5040 *
               5050 *---CLOSE IOB--------------------
               5060 SLIOB.CLOSE
1332- 01       5070        .DA #1
1333- 00       5080        .HS 00       RefNum
               5090 *
               5100 *---SETMARK IOB------------------
               5110 SLIOB.SETMARK
1334- 02       5120        .DA #2
1335- 00       5130        .HS 00       RefNum
1336- 00 00 00 5140        .HS 00.00.00
               5150 *
               5160 *---READ IOB---------------------
               5170 SLIOB.READ.INDEX
1339- 04       5180        .DA #4
133A- 00       5190        .HS 00       RefNum
133B- 00 09 8C
133E- 00 00 00 5200        .DA BUF.900,$008C,$0000
               5210 *--------------------------------
               5220        .PH $13C1
13C1-          5230 MSG..1 >MSG "Place the AppleWorks PROGRAM disk in Drive 1 and press Return.  "
               5240        .EP
               5250 *--------------------------------
               5260        .PH $1AFC
               5270 * (1AFC) 1042 12B9 24C6 270D 272E 278F 2B2E 2CF3
               5280 * (1B00) 1045 1265 23F1 249F 26E4 2771 2960 2CEA
               5290 SET.PRODOS.BITMAP
1AFC- A9 FF    5300        LDA #$FF     Protect all main RAM from ProDOS
1AFE- D0 02    5310        BNE SC.BM    ...always
               5320 CLR.PRODOS.BITMAP
1B00- A9 00    5330        LDA #$00     De-protect all main RAM
1B02- A2 13    5340 SC.BM  LDX #$13     Affects RAM from $0800 thru $9FFF
1B04- 9D 58 BF 5350 .1     STA PRODOS.BITMAP,X
1B07- CA       5360        DEX
1B08- D0 FA    5370        BNE .1
1B0A- 60       5380        RTS
               5390        .EP
               5400 *--------------------------------
```

## Dissecting AppleWorks SEG.M0 and SEG.M1 Files...

...Bob Sander-Cederlof

The AppleWorks Program disk contains two files of type $00, called SEG.M0 and SEG.M1. These contain the actual code for the three applications (data base, word processing, and spreadsheet) as a series of overlay segments.

These two files could really be just one, and I don't understand why they are not. It takes extra logic inside AppleWorks to manage them as two files, and the code would be a little simpler if there were only one. As it is, the first nine of 43 overlays are on SEG.M0, and the remaining 34 are on SEG.M1.

The overlay loader first decides which file to open, and then reads in the first 140 bytes of that file. There is a "directory" of sorts at the beginning of each file: one 3-byte value for each segment in the file. The 3-byte value is the offset within the file where the overlay begins. For example, looking at the beginning of the SEG.M0 file for AppleWorks version 1.3, in 3-byte groups, I see:

```
        00 00 00
        21 00 00
        4C 30 00
        9A 35 00
        and so on
```

This means that overlay segment 0, which does not really exist, begins at offset $000000 in the file. Overlay segment 1 begins at $000021, or with the 33rd byte of the file. Segment 2 begins at $034C in the file, and so on. By subtracting the beginning address of the segment I want to load from the address of the begining of the next segment I get the number of bytes in the desired segment.

I decided to write a program to analyze these file headers for me, and print out a list of file offsets and lengths for each segment. The program follows, but before describing it I need to mention a table inside APLWORKS.SYSTEM. A table I call "SEGMENT.TABLE" begins at $10A7 (in version 1.3). There are four bytes for each segment in this table. The first of each group of four is the page number where the corresponding segment should be loaded. Entries for segments $0F, $14, and $15 are zero, meaning these segments do not exist. Segment $00 does not exist either, but it is eliminated by other means. The other three bytes of each 4-byte group are used by the overlay loader to keep track of which overlay segments are already in RAM, in AuxRAM, or in a RamFactor type card.

I decided to copy the loading page numbers from this table into my little analysis program, so that it could also print out the loading address for each segment. You can see my copy at lines 1860-1930. I have added three labels to indicate which overlay segments belong to which of the three applications. The Data Base code is in segments $01 through $0F, the Word Processor in segments $10-17, and the SpreadSheet in segments $18-2B. At least that is what I think is true, someone correct me if you know better.

My analysis program has several interesting wrinkles, of interest beyond the overall function of the program. I have defined two useful macros in lines 1070-1180. The first one, PRINT, generates a JSR PRINT followed by a zero-terminated string. As long as the string is purely printable ASCII characters which will fit in a .AS directive, the macro works nicely. The PRINT subroutine in lines 1680-1840 picks up the string which follows the JSR PRINT and prints it out, modifies the return address, and returns to the next instruction following the string.

The second macro calls on the Apple monitor to print a byte in hexadecimal. If there is no parameter in the >HEX macro call line, the macro will assume the byte to be printed is already in the A-register and generate only a JSR PRBYTE line. However, if you include one parameter, the macro will generate a LDA instruction to load the value into the A-register first. For example:

```
        >HEX             generates  JSR PRBYTE

        >HEX #$24        generates  LDA #$24
                                    JSR PRBYTE

        >HEX BUFFER      generates  LDA BUFFER
                                    JSR PRBYTE

        >HEX "BUF+2,Y"   generates  LDA BUF+2,Y
                                    JSR PRBYTE
```

Note that in the last example I had to put the BUF+2,Y in quotation marks, so that the assembler would include the ",Y" as part of the ]1 parameter.

When you use >HEX with no parameter, you also must not put a comment on the line. The first word of any comment would be considered as a parameter, generating bad results.

I assembled the program with the .LIST MOFF option, so that macro expansions are not shown.

The program assumes that you have BLOADed the SEG.Mx file header into a buffer starting at $0A00. On my disk I have those files in a subdirectory called AW, so after assembling the program I typed:

```
        BLOAD AW/SEG.M0,T$00,A$A00,L200
        MGO T
        BLOAD AW/SEG.M1,T$00,A$A00,L200
        MGO T
```

Lines 1200-1280 search through the index beginning at $0A00 for the first 3-byte entry which is not all zero. There is one 000000 entry in the SEG.M0 file, and there are ten of them in the SEG.M1 file. The first non-zero entry actually also points just past the end of the index header, so I save that value for a loop termination count in lines 1290-1310.

Lines 1330-1350 print "SEGMENT." and the two digit segment number in hexadecimal. Lines 1360-1390 decide whether the segment exists or not, and prints ": null" if it does not. If the segment does exist, lines 1410-1560 print out the "A$xx00" load address, using the value from my LOAD.ADDRESS.TABLE; the "B$xxxxxx" file offset; and the "L$xxxx" segment length.

Lines 1570-1630 advance to the next three byte entry, and loop back if not at the end of the index header.

Using the information that prints out I could easily load any individual segment from within one of the SEG.Mx files and save it on its own private BIN file. For example, to load and save segment $20 I would type:

```
       BLOAD SEG.M1,T$00,A$1000,B$0144BB,L$1E4E
       BSAVE SEGMENT.20,A$1000,L$1E4E
```

I could modify the analysis program to generate an EXEC file which would include two lines like these for each existing segment. Then EXECing the file would automatically produce 40 separate binary files, one for each overlay segment (not 43, because there are three "null" segments). This would make it easier to disassemble each one. I probably will end up modifying it this way eventually.

Another interesting program would create a new SEG.Mx file from a set of separate binary files within a subdirectory. What do you bet Robert Lissner has just such a program?

```
               *SAVE S.SHOW.INDEX
               *--------------------------------
FD8E-          CROUT  .EQ $FD8E
FDDA-          PRBYTE .EQ $FDDA
FDED-          COUT   .EQ $FDED
               *--------------------------------
                      .MA PRINT
                      JSR PRINT    Print 00-term'd string after
                      .AS -"]1"    here is the string
                      .HS 00       here is the 00-terminator
                      .EM
               *--------------------------------
                      .MA HEX
                      .DO ]#       If any parameter, LDA it
                      LDA ]1       here is the LDA
                      .FIN
                      JSR PRBYTE   Print <A> in hexadecimal
                      .EM
               *--------------------------------
               T
0800- A2 FF           LDX #-1      X will be the segment number
0802- A0 FD           LDY #-3      FIND FIRST NON-ZERO ENTRY
0804- C8       .1     INY
0805- C8              INY
0806- C8              INY
0807- E8              INX
0808- B9 00 0A        LDA BUF,Y
080B- F0 F7           BEQ .1       ...this entry was empty
080D- 38              SEC
080E- E9 03           SBC #3
0810- 8D 8B 08        STA Y.LIMIT
               *--------------------------------
0813-          .2   >PRINT "SEGMENT."
081F- 8A              TXA          PRINT SEGMENT NUMBER
0820-                >HEX
0823- BD B0 08        LDA LOAD.ADDRESS.TABLE,X
0826- D0 0E           BNE .3       not a null segment
0828-               >PRINT ": null"
0833- 4C 7E 08        JMP .4
               *--------------------------------
0836-          .3   >PRINT ": A$"       Print load address
083F-                >HEX "LOAD.ADDRESS.TABLE,X
0845-               >PRINT "00, B$"     Print file offset
084F-                >HEX "BUF+2,Y
0855-                >HEX "BUF+1,Y
085B-                >HEX "BUF,Y
0861-               >PRINT ", L$"       Print segment length
0869- 38              SEC
086A- B9 03 0A        LDA BUF+3,Y
086D- F9 00 0A        SBC BUF,Y
0870- 48              PHA
0871- B9 04 0A        LDA BUF+4,Y
0874- F9 01 0A        SBC BUF+1,Y
0877-                >HEX
087A- 68              PLA
087B-                >HEX
087E- 20 8E FD .4     JSR CROUT         Next line
0881- E8              INX               Next segment number
0882- C8              INY               Next header pointer
0883- C8              INY
0884- C8              INY
0885- CC 8B 08        CPY Y.LIMIT       Into first segment?
0888- 90 89           BCC .2            ...no, more lines
088A- 60              RTS               ...done
               *--------------------------------
088B-          Y.LIMIT    .BS 1
               *--------------------------------
               PRINT
088C- 68              PLA          POP RETURN ADDRESS
088D- 8D 9D 08        STA .2+1     BECAUSE IT POINTS TO STRING
0890- 68              PLA
0891- 8D 9E 08        STA .2+2
0894- EE 9D 08 .1     INC .2+1     BUMP POINTER TO NEXT CHAR
0897- D0 03           BNE .2
0899- EE 9D 08        INC .2+1
089C- AD 33 33 .2     LDA $3333    GET NEXT CHAR OF STRING
089F- F0 06           BEQ .3       00 = END OF STRING
08A1- 20 ED FD        JSR COUT     PRINT CHAR
08A4- 4C 94 08        JMP .1       ...NEXT
```

```
08A7- AD 9E 08 1800 .3      LDA .2+2      PUT RETURN ADDRESS ON STACK
08AA- 48       1810         PHA
08AB- AD 9D 08 1820         LDA .2+1
08AE- 48       1830         PHA
08AF- 60       1840         RTS
               1850 *-------------------------------
               1860 LOAD.ADDRESS.TABLE
08B0- 30       1870             .HS 30    dummy entry for segment 00
08B1- 35 45 45
08B4- 45 45 45
08B7- 4E 4E 4A 1880 DATA.BASE .HS 35.45.45.45.45.45.4E.4E.4A
08BA- 4E 4E 4E
08BD- 53 4E 00 1890             .HS 4E.4E.4E.53.4E.00
08C0- 35 3D 3D
08C3- 40 00 00
08C6- 67 77    1900 WORD.PROC .HS 35.3D.3D.40.00.00.67.77
08C8- 33 3B 53
08CB- 53 53 53
08CE- 53 53 45 1910 SPRD.SHEE .HS 33.3B.53.53.53.53.53.53.45
08D1- 64 64 64
08D4- 64 64 64
08D7- 64 64 64 1920             .HS 64.64.64.64.64.64.64.64.64
08DA- 64 64    1930             .HS 64.64
               1940 *-------------------------------
0A00-          1950 BUF     .EQ $A00
               1960 *-------------------------------
```

## Backup/Restore for RamFactor DOS Partition

...Bob Sander-Cederlof

The Applied Engineering RamFactor memory card is now widely distributed, and with good reason. It is among my very favorite cards, and I use it heavily every day. I use mine with the RamCharger battery backup system, so that the memory stays loaded and ready all the time.

I have mine partitioned into two parts: a DOS 3.3 partition of 140K (floppy size), and a ProDOS partition with all the rest. I have been using Copy II Plus to backup the ProDOS partition on a 3.5 inch disk, but I haven't been keeping an up-to-date backup of the DOS partition. (Copy II Plus cannot access the DOS partition, or at least not when I have just loaded Copy II Plus out of the ProDOS partition on the same card.)

I have sometimes used FID to copy every file from the DOS partition to a floppy, but that takes a long time. In fact, when I tried it today, it took 160 seconds to save 31 files. And that only backs up the files. If the RamFactor is somehow clobbered, I will also need to restore the DOS image. My DOS is significantly patched, so I would really like to have it on the floppy too. Let's see.... I could boot from the RamFactor, go into Applesoft, load the HELLO program, directly type in the INIT command to initialize a floppy, then go into FID and copy all the rest of the files. Too much! And anyway, how do I get the floppy's contents copied back into the RamFactor?

Looking at the whole problem another way, what if I did not have the RamCharger? Then I would need to copy all the files and a DOS image into the card at least once a day. That could be really tiresome.

I decided to write a program to simplify things. My program has three parts: Format, Backup, and Restore. FORMAT.FLOPPY will do a raw format of a floppy disk. That is, it only writes the sector headers for 16 sectors on each of 35 tracks; it does not write a DOS image or an empty catalog. If the floppy I want to use has already been formatted, I can skip using FORMAT.FLOPPY.

BACKUP copies all the sectors of all 35 tracks from the RamFactor to the Floppy, and RESTORE does the reverse. BACKUP takes 46 seconds to copy and verify all 560 sectors, and RESTORE takes 25 seconds to read them back. Not as fast as possible, considering how fast Locksmith can copy one un-protected floppy to another, but my program is considerably shorter than Locksmith. If I leave out the Verify phase, BACKUP takes only 25 seconds.

Since all sector I/O is done by calls to RWTS, this same program could be used to backup and restore floppy-sized volumes on the Sider Hard Disk, with only minor modifications. Sider comes with a modified version of FID which already can do an "image" copy, as they call it, but it is too slow for me.

Another set of modifications would make my program work with 400K DOS partitions on the RamFactor and 3.5 inch disks formatted for use with DOS.

I decided to keep the program simple, for now. The slot numbers for the floppy drive and the RamFactor are assembled in at lines 1020-1030. A fancy program would probably do a slot search to find them, and give you a choice if there were more than one of either in the computer. A fancy program would also give you a little menu for selecting FORMAT, BACKUP, or RESTORE; I didn't do that either, but you can easily add one. One more improvement would be to automatically format the floppy if it isn't already formatted.

Well, let's look at what I DID do! Lines 1300-1370 show the two entry points for BACKUP and RESTORE. They load the slot numbers (shifted into the high nybble, so we say it is slot*16) into the A- and X-registers and go to COPY. COPY copies 35 tracks of 16 sectors each from the slot in the A-register to the slot in the X-register.

Lines 1390-1790 are the COPY subroutine. As each track is copied, I display the track number in hex, and the letters R, W, and V on the screen. After the letter R is displayed, I read the entire track from the source slot/drive into a buffer which starts at $2000. After the letter W is displayed, I write out the track to the destination slot/drive. After the letter V is displayed, I read the entire track again, this time from the destination slot/drive. If RWTS does not report any error, I assume the track is good. A more excellent way might be to read the destination track into a different buffer, and compare all 4096 bytes with the original.

After the track is verified I print two more spaces, making the total number of characters displayed for each track, eight. This means I display either 5 or 10 tracks on a screen line, depending on whether the screen is set to 40- or 80-columns.

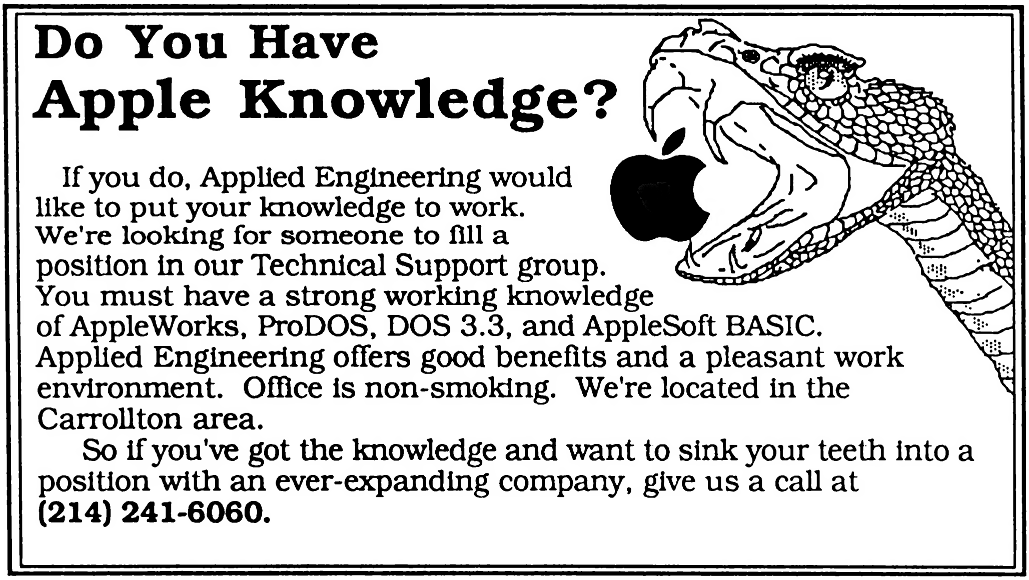
If you want to skip the verify step altogether, you can delete lines 1620-1660 and add one more JSR COUT after line 1700.

Reading or writing a track is handled by lines 1840-2070, RW.TRACK. This finishes setting up the IOB and calls RWTS to do the I/O. If RWTS reports any error during the copy process, copying stops and I print all the interesting information about the error. Lines 2090-2280 do the printing. You can also stop a copy by typing any key. Lines 1710-1720 look for a keypress after finishing each track, and abort if one is found.

RW.TRACK reads the sectors in the order 15 down to 0. Of course, RWTS translates the logical sector numbers into "real" sector numbers, but we don't need to worry about that. There is an optimum order to read or write sectors, and it depends on several factors. First, it depends on the interleaving order on the disk, as viewed through the RWTS logical sector numbers. Second, it depends on how much time is wasted between reading or writing each sector. Programs like Locksmith read an entire track in one revolution of the disk, once the beginning of any sector is found. Locksmith also writes an entire track in one revolution. Using RWTS that is not possible, but you can probably do it in an average of 2.5 revolutions if you are smart enough. The drive spins at 5 revolutions per second, by the way.

I tried reading the sectors in the order 0 to 15, and it took 100 seconds just to READ 35 tracks. In the order 15 to 0, this took only 24 seconds. The disk turns 120 times in 24 seconds, so I am averaging less than 3.5 revolutions per track including the time it takes to step from track to track. (Theodore Roosevelt used to warn national leaders around the world about the hazards and long-term negative results of a habit of revolution, but I think he was on a different track.)

If you decide to type in this program, with or without any modifications, be very careful about using it. You can easily wipe out the contents of the RamFactor with only a tiny bug. I carefully made a backup with FID before testing RESTORE. It turned out I didn't need it, but I am still glad I did.



```
                  1000 *SAVE S.FAST RAMFACTOR BACKUP
                  1010 *--------------------------------
04-               1020 RFSLOT .EQ 4          RAMFACTOR SLOT
06-               1030 FLSLOT .EQ 6          FLOPPY SLOT
                  1040 *--------------------------------
03E3-             1050 GETIOB .EQ $3E3
03D9-             1060 RWTS   .EQ $3D9
                  1070 *--------------------------------
C000-             1080 KEYBOARD   .EQ $C000
C010-             1090 STROBE     .EQ $C010
                  1100 *--------------------------------
FD8E-             1110 CROUT  .EQ $FD8E
FDDA-             1120 PRBYTE .EQ $FDDA
FDED-             1130 COUT   .EQ $FDED
                  1140 *--------------------------------
                  1150        .DUMMY
                  1160        .OR $B7E8
B7E8-             1170 IOB    .BS 1          Reference "Beneath Apple DOS"
B7E9-             1180 SLOT16 .BS 1
B7EA-             1190 DRIVE  .BS 1
B7EB-             1200 VOLUME .BS 1
B7EC-             1210 TRACK  .BS 1
B7ED-             1220 SECTOR .BS 1
B7EE-             1230        .BS 2          ADDR OF DCT
B7F0-             1240 BUFADR .BS 2
B7F2-             1250        .BS 2
B7F4-             1260 CMD    .BS 1
B7F5-             1270 ERRCOD .BS 1
                  1280        .ED
                  1290 *--------------------------------
                  1300 BACKUP
0800- A9 40       1310        LDA #RFSLOT*16     From RamFactor to Floppy
0802- A2 60       1320        LDX #FLSLOT*16
0804- 4C 0B 08    1330        JMP COPY
                  1340 *--------------------------------
                  1350 RESTORE
0807- A9 60       1360        LDA #FLSLOT*16     From Floppy to RamFactor
0809- A2 40       1370        LDX #RFSLOT*16
                  1380 *--------------------------------
080B- 8D 61 08    1390 COPY   STA SRC.SLOT16     Save Source Slot*16
080E- 8E 62 08    1400        STX DES.SLOT16     Save Destination Slot*16
0811- A9 01       1410        LDA #1             Both are drive 1
0813- 8D EA B7    1420        STA DRIVE
0816- A0 00       1430        LDY #0             For Track = 0 to 34
0818- 8C EC B7    1440 .1     STY TRACK
081B- 98          1450        TYA
081C- 20 DA FD    1460        JSR PRBYTE         Print track number in hex
081F- A9 AD       1470        LDA #"-"           and a dash...
0821- 20 ED FD    1480        JSR COUT
                  1490 *---READ TRACK------------------
0824- A9 D2       1500        LDA #"R"           Print "R"
0826- AE 61 08    1510        LDX SRC.SLOT16     Read track from Source Slot
0829- A0 01       1520        LDY #1        READ COMMAND
082B- 20 63 08    1530        JSR RW.TRACK
082E- B0 60       1540        BCS RWTS.ERROR     ...Error
                  1550 *---WRITE TRACK ON FLOPPY--------
0830- A9 D7       1560        LDA #"W"           Print "W"
0832- AE 62 08    1570        LDX DES.SLOT16     Write track to Dest. Slot
0835- A0 02       1580        LDY #2        WRITE COMMAND
0837- 20 63 08    1590        JSR RW.TRACK
083A- B0 54       1600        BCS RWTS.ERROR     ...Error
                  1610 *---VERIFY TRACK ON FLOPPY-------
083C- A9 D6       1620        LDA #"V"           Print "V"
083E- AE 62 08    1630        LDX DES.SLOT16     Read track for Dest. Slot
0841- A0 01       1640        LDY #1        READ COMMAND
0843- 20 63 08    1650        JSR RW.TRACK
0846- B0 48       1660        BCS RWTS.ERROR     ...Error
                  1670 *---CHECK FOR ABORT--------------
0848- A9 A0       1680        LDA #" "           Print 2 blanks
084A- 20 ED FD    1690        JSR COUT           allowing 8 screen columns per track
084D- 20 ED FD    1700        JSR COUT
0850- AD 00 C0    1710        LDA KEYBOARD       Press any key to abort
0853- 30 08       1720        BMI .2             ...ABORT
                  1730 *---NEXT TRACK------------------
0855- AC EC B7    1740        LDY TRACK
0858- C8          1750        INY
0859- C0 23       1760        CPY #35       limit to 35 tracks
085B- 90 BB       1770        BCC .1        ...more to do
085D- 8D 10 C0    1780 .2     STA STROBE    ...done
0860- 60          1790        RTS
```

```
                *-------------------------------
0861-           SRC.SLOT16 .BS 1
0862-           DES.SLOT16 .BS 1
2000-           TRKBUF .EQ $2000 ... 2FFF
                *-------------------------------
                *    (A)=CHAR TO BE PRINTED
                *    (X)=SLOT*16
                *    (Y)=1 for READ, 2 for WRITE
                *-------------------------------
                RW.TRACK
0863- 8E E9 B7         STX SLOT16   Save slot*16 in IOB
0866- 8C F4 B7         STY CMD      Save command in IOB
0869- 20 ED FD         JSR COUT     Print R, W, or V
086C- A9 00            LDA #0
086E- 8D EB B7         STA VOLUME   Accept any volume number
0871- 8D F0 B7         STA BUFADR   Lo-byte of buffer address=00
0874- A9 20            LDA /TRKBUF  Hi-byte of buffer address
0876- 8D F1 B7         STA BUFADR+1
0879- A0 0F            LDY #15      For Sector = 15 to 0 step -1
087B- 8C ED B7  .1     STY SECTOR
087E- 20 E3 03         JSR GETIOB   Setup for RWTS Call
0881- 20 D9 03         JSR RWTS
0884- B0 09            BCS .2       ...ERROR
0886- EE F1 B7         INC BUFADR+1 Next Buffer Address
0889- AC ED B7         LDY SECTOR
088C- 88               DEY          Next Sector
088D- 10 EC            BPL .1       ...more to do
088F- 60        .2     RTS          ...done
                *-------------------------------
                RWTS.ERROR
0890- 20 8E FD         JSR CROUT    Print all that's of interest
0893- AD E9 B7         LDA SLOT16   Slot * 16
0896- 20 BA 08         JSR PRBYTE.SPACE
0899- AD EA B7         LDA DRIVE    Drive number
089C- 20 BA 08         JSR PRBYTE.SPACE
089F- AD EC B7         LDA TRACK    Track number
08A2- 20 BA 08         JSR PRBYTE.SPACE
08A5- AD ED B7         LDA SECTOR   Sector number
08A8- 20 BA 08         JSR PRBYTE.SPACE
08AB- AD F4 B7         LDA CMD      Command Code
08AE- 20 BA 08         JSR PRBYTE.SPACE
08B1- AD F5 B7         LDA ERRCOD   Error Code
08B4- 20 BA 08         JSR PRBYTE.SPACE
08B7- 4C 8E FD         JMP CROUT
                *-------------------------------
                PRBYTE.SPACE
08BA- 20 DA FD         JSR PRBYTE   Print value in hex
08BD- A9 A0            LDA #" "     and a space
08BF- 4C ED FD         JMP COUT
                *-------------------------------
                FORMAT.FLOPPY
08C2- A9 01            LDA #1
08C4- 8D EB B7         STA VOLUME   Make it volume 1 (why not?)
08C7- 8D EA B7         STA DRIVE    on Drive 1
08CA- A9 60            LDA #FLSLOT*16    Do it to the floppy
08CC- 8D E9 B7         STA SLOT16
08CF- A9 04            LDA #4       FORMAT COMMAND Code
08D1- 8D F4 B7         STA CMD
08D4- 20 E3 03         JSR GETIOB   Set up RWTS call
08D7- 20 D9 03         JSR RWTS
08DA- B0 B4            BCS RWTS.ERROR
08DC- 60               RTS          Done
                *-------------------------------
```

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $24 per year in the USA, Canada, and Mexico, or $36 in other countries. Back issues are $1.80 each for Volumes 1-7, $2.40 each from later Volumes (plus postage). A subscription to the newsletter with a Monthly Disk containing all program source code and article text is $64 per year in the USA, Canada and Mexico, and $90 to other countries.